Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures

Françoise André, Erwan Daubert, Guillaume Gauvrit

2010 Fifth International Conference on Internet and Web Applications and Services

Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures

Françoise André, Erwan Daubert, Guillaume Gauvrit
IRISA / INRIA
Campus de Beaulieu, 35042 Rennes cedex, France
{Francoise.Andre, Erwan.Daubert, Guillaume.Gauvrit}@irisa.fr

***Abstract*—Many software applications are now built from services which run on heterogeneous platforms and are accessed by several users. These environments are in constant evolution. So applications and services have to dynamically adapt in order to satisfy the quality of service required by the users. Programming adaptation facilities taking into account the different situations that could happen is a difficult task. Our objective is to provide mechanisms for self-adaptation of services. We propose a generic framework that allows to specify different kinds of adaptation, in various environments. This paper presents the overall framework and details some of its functionalities: the monitoring, the decision to adapt and the planning of adaptation actions. The current state of our implementation for an OSGi platform is described.**

***Keywords*-Self-adaptation; SOA; Framework for Dynamic Adaptation; Cloud Computing;**

I. Introduction

Applications are more and more build as a composition of services running on large scale, dynamic and heterogeneous environments. A Service Based Application (SBA) is a software architecture where the basic element is the service, performing a single functionality, such as accessing a resource or managing a data structure. The services communicate with each others to satisfy the objective of the application, by using communication protocols through well-defined interfaces such as SOAP for web services. The very large scale of the execution environment is to be measured in terms of number of users, service-oriented platforms and computers. The dynamism results from node volatility (due to failures, maintenance, voluntary connections or disconnections), service evolution (new services may be deployed, existing ones be removed) and varying users demands requiring different levels of quality of service. All these factors lead to the necessity for dynamic (i.e. at run time) service, application and infrastructure adaptation, without human intervention.

Our objective is to define a generic context-aware framework to build self-organizing service-oriented architectures for cloud computing.

Section II briefly places our work relatively to related works. Next, Section III presents an overview of the framework. Then the different main functions of our framework are detailed, starting with the monitoring function in Section IV. Following is the decision and analysis in Section V. Then the planning is described in Section VI. Finally the execution phase of the adaptation is presented with the example of our implementation for the migration of a service in Section VII. Section VIII concludes this article and presents our future works on the framework.

II. State of the Art

Currently, the works done on dynamic self-adaptation are focused on component-based software architectures, for which multiple frameworks have been developed, such as Dynaco [1] and SAFRAN [2]. Both of them are generic, i.e. they separate the adaptation phases and let free the choice of the adaptation logic. The MAPE model [3] (Monitoring, Analysis, Planning, Execution) is now a standard reference for specifying a way to divide the different adaptation phases.

Various works proposing means for dynamic adaptation have been done in specific contexts, either restricted to an application domain or to a given infrastructure [4], [5], [6], [7]. Very few are based on a generic framework and enable self-adaptation.

III. Framework

For some years, the concepts of autonomic and adaptive applications has grown in importance in several research areas, such as mobile computing, grid computing and enterprise computing. Applications adapt at run-time for a range of different purposes, including to cope with the amount of available resources, to use a more efficient program and to better satisfy the user needs. But introducing facilities for adaptation in exiting code is a very difficult task. Even when designing new applications, taking into account the various situations where adaptation may occur is almost impossible for the designer. Ad hoc solutions are not a good way to solve the problem. The use of a generic framework, built separately from the functional code, specifically dedicated to run-time adaptation, as described by the model MAPE in [3] and capable to itself evolve, seems to be the only solution for the long time.

Based on our previous experience aiming at designing a framework for adaptation of a component based applica-

978-0-7695-4022-1/10 $26.00 © 2010 IEEE
DOI 10.1109/ICIW.2010.52

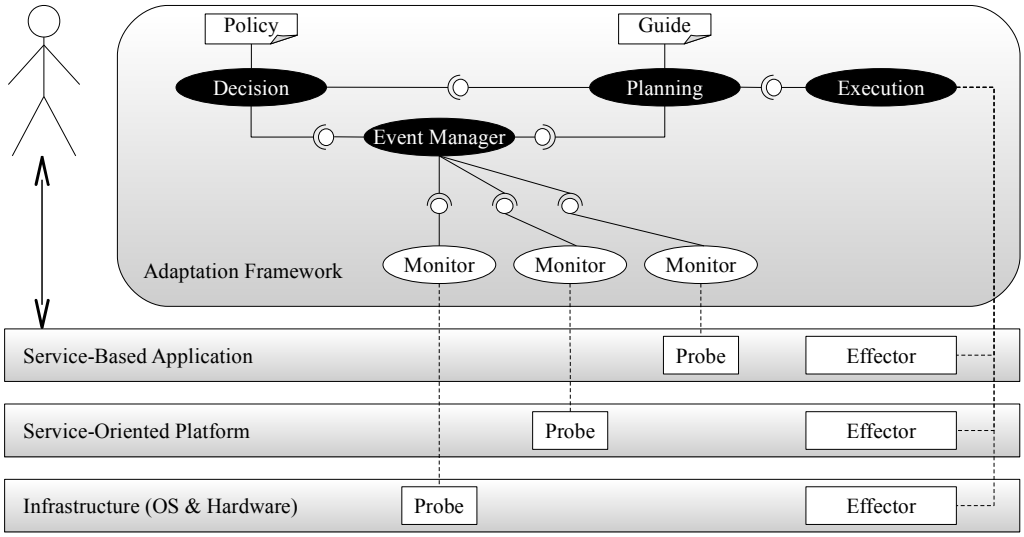

Figure 1. Framework

tion [8], we propose an extension taking into account the world of distributed services on heterogeneous platforms such as OSGi [9], SCA [10] and web services [11]. The framework allows to specify any kind of adaptation by a precise decomposition of the different functionalities that may be involved in an adaptation process. Each of these functionalities may be specialized to fit some particular need. Moreover the separation of the different adaptation functions gives a clear and well structured code, easier to maintain and to evolve if new situations appear.

Five major types of adaptation can be done using our framework on a SBA: parametric adaptation modifies the value of an existing parameter of a service, for example it modifies the bitrate of a video stream; functional adaptation replaces one function implementation by another, leaving the interfaces unchanged, for example it changes the encoding algorithm for a video; behavioural adaptation changes the manner a service acts and possibly its interfaces, for instance it changes a visual representation by a vocal one; structural adaptation modifies a composition of services inside an application, for instance adding a new service for security purpose of the application; environmental adaptation allows to change the outside world of the application, for example it migrates a service for a better use of the underlying physical infrastructure.

As can be deduced from this description of the adaptation types, our framework works at different levels, ranging from a single service, a composition of services in one application, to several applications running on heterogeneous service oriented platforms and distributed computers. Concerning this last point, to cope with the distributed aspect, our framework may be itself distributed. This enforces its scalability. Meanwhile in that case the framework should remain coherent; this property is ensured by providing cooperation and coordination mechanisms, as we will see in the next sections.

To finish this general description of the framework, it is divided into the four main functionalities of the *MAPE* model: Monitoring or observation, Analysis or decision, Planning and Execution. The observation function is realized by a set of probes, monitors and event managers, and triggers the adaptation system depending on the state of the environment and the services (see Section IV); the analysis and decision function decides of an adaptation strategy and will be described in Section V; the planning function has to define a set of actions that will satisfy the strategy chosen by the decision phase and schedules them (see Section VI); the last step is the execution of the concrete actions on the different elements that are impacted by the adaptation (see Section VII).

The figure 1 resumes the main points of our proposal:

- It is multilevel: on the top we find the user who, by changing his needs, may trigger an adaptation; on the bottom the infrastructure (OS and material) may also requires adaptation due to nodes volatility or changes in the OS policies; in the middle the services, their composition in SBA and the associated Service Oriented Platforms where all the different types of adaptation are performed using our framework.
- Monitoring is done at each level and reciprocally adaptation actions may also be performed at different levels, depending on the adaptation types.
- The different functions that compose the adaptation framework are shown, together with the means to specialize them by decision policy and planning guide, as it will be described in the following sections.
- In order not to overload the picture, the distributed aspect of our system doesn't appear explicitly in the

figure, but all the elements shown may be replicated on different nodes to cope with large scale environments.

IV. Monitoring

The monitoring function of our framework is used to collect an informative and dynamic view of the adaptive entity and its environment. This view should include every pieces of information deemed useful to take an adaptation decision, including information on the hardware, the operating system and the service-oriented platform the adaptive service is running on, in addition to the adaptive application and the services themselves. This dynamic view is pictured by values representing states of the system. Events, coming from the observed elements modify those states.

More precisely, the monitoring function is designed to collect relevant events in order to picture a low-level view of all the elements. Such a view is useful to have precise information but is not very well suited to get a global representation of states of the system. To do so, the monitoring function of our framework can compose events to represent derivative values better suited to picture a higher-level, synthetic view of the system.

In addition to listening to events occurring in the system, the monitoring function can also actively probe periodically selected values to generate events and update the view. Events are transmitted to the decision function only when differing from previous values. The context view is built using an EQL-based model and implemented using the WildCAT framework [12].

In our framework, the monitoring phase of the MAPE model is divided into multiple services and elements. Ad hoc *probes* are the software or hardware elements taking the measures needed to create the events. The probes are listened to and queried by *monitors*, which are services providing events using the same interface to the *event manager*. Monitors constitute an abstraction layer of the ad hoc probes. The event manager is the service collecting the events, composing them and keeping a view of the system.

The probes can take measures in every layer of the system. At the operating system level, a probe can for example measure the state of hardware resources, such as the available RAM or the number of processors. The procfs virtual file system under Unix-like OSs (usually mounted at `/proc`) is an example of an OS level probe. At the service platform level, probes can notice that a service appears or disappears. Whereas in the adaptive services, probes can keep under observation various evolutions, such as the size of a buffer or calls to functions.

Since probes can use various communication interfaces, monitors interfaces them to provide events under the same service-oriented interface. A single monitor can listen multiple probes and push multiple events. Monitors have to be able to answer to requests on the value for the events they monitor.

The event manager is designed to collect and compose events coming from monitors, constituting composite events. It can also make computations on an event over a window of time. For example, a composite event can be the average number of requests being processed by processors. A composite event can also be the maximum time to process a request over the last minute. Those computations are described in a dedicated language. Since the framework is distributed, there can be multiple event managers spread over various service-oriented platforms. The event manager also checks if updated values of event are different from previous ones and only transfers them to the decision service if it is the case. The decision and planning functions of the framework can request values of events through the event manager, in order for them to get complementary information to execute their tasks when needed.

V. Decision

The analysis function of the MAPE model is designed to take adaptation decisions. When a change arises in the monitored elements, the monitoring function sends a notice to the decision function. Upon receiving a new notice, the decision function has to analyse and decide if an adaptation is needed and to choose an adaptation strategy if needed.

The decision function of our framework is constituted by many distributed decision services, each of them made up by a negotiator handling the communication and possibly two decision engines. In an application composed of distributed services, such as web services in clouds, taking a coherent and global view of every elements contributing to the application is next to impossible, especially in a timely fashion. Thus, in our framework the decision function is composed of various *decision services* cooperating together to take adaptation decisions, each decision service being in charge of a subset of the application. A decision taken by the decision function is reified into multiple strategies, each one being a part of a distributed strategy.

Distributed strategies are made the following way. When a decision service notices a change in its view of the system that necessitates an adaptation, the decision service issues a strategy stating what are the changes to make. If the application of the strategy doesn't need involvement from other decision services, the strategy is handled to the local planning function. In the other cases, the parts of the strategy needing involvement from other adaptation services will be negotiated with the concerned decision services. Those negotiations may end up initiating other strategies that, together with the original strategy, will be constituting a distributed strategy.

The cooperation between decision services is handled by negotiation mechanisms using for instance FIPA protocols [13]. As a consequence a decision service is divided into two parts: the decision-maker and the negotiator, as shown in Figure 2. The role of the negotiation is to make the various

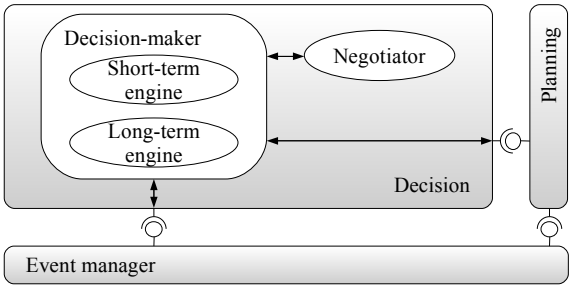

Figure 2. A decision service

decisions services agree on common parts of distributed strategies. A decision service can submit a strategy to another decision service. This one has to analyse what is involved and then to decide either to apply it, reject it or make a counter-proposal. Indeed, a service can depend on many things unknown to the decision service initiating a distributed strategy negotiation; those dependencies make the application of the strategy not always as straightforward as it could appear to the issuer of the original strategy. Once a distributed strategy is negotiated, strategies are given to the relevant services of the planning function.

Moreover, our framework offers the possibility for the decision-maker to be decomposed into two decision engines running concurrently. These two engines are complementary: the first one focus on short-term decisions and the second on long-term ones. Both are used to make adaptation decisions, that is strategies. However they don't reason the same way and corresponds to different adaptation needs. The short-term engine makes fast and simple decisions and is useful when rapid reactions are needed. The other is slower to analyse situations but can make complex strategies and optimize the application for the long-term.

The short-term engine may for instance use a rule-based engine compliant to the Java Rule Engine API (JSR-94), such as JESS or JRuleEngine. A rule-based engine can be seen as an elaborated parser of *if-then* statements. It is used to make relatively fast and simple decisions, in order to enforce a valid state of the distributed application. Since this engine can only make strategies that are defined statically, it is rather limited in a very heterogeneous and dynamic environment. It can be used for example to decide that when a service fails and an equivalent one can be found, this service replace the failed one.

The long-term engine may use an engine based on utility functions [14]. Utility functions are functions computing the *utility* of a configuration of a system, enabling to compare configurations. A configuration found with an utility significantly higher than the utility of the configuration currently in use can be used to make a strategy. Since it can create strategies dynamically, it is well suited for an use in a heterogeneous and dynamic environment. In addition, since its strategies are made to fulfil a goal that can change dynamically, it is well suited in a context where goals of an application can be modified by SLAs. It's more an engine optimizing the current state of the system than an engine reacting to events. Thus it can optimize the application or services following the constraints of the developers (for example to limit the consumption of resources) and the constraints of the users (for example to lower the response time), making compromises. However, this engine is not as fast as an rule-based engine and might not be able to make a strategy in a timely fashion.

VI. Planning

As seen in the previous section, a decision result is represented by a strategy. A strategy defines the new state to reach but it doesn't define how to obtain this state. This is the role of planning phase. To do so, the planner looks for a set of actions that are able to turn the current state of the system (here meaning application and services) into the new state.

Various types (parametric, functional, behavioural, structural, environmental) of actions can be involved to adapt a system, as described in Section III. Each type of adaptation corresponds to a set of actions that can be used to implement it. The role of the planner is to select the appropriate actions for the current situation. In most of the cases of adaptation, several actions are needed to implement a strategy. For example the structural adaptation which consists to add a new service in an application (i.e. a composition of services) may be implemented by choosing the new service, connecting it to one or several services of the old composition, after having removed their previous links. In such situation a scheduling policy should be specified by the planner. Indeed some actions are dependant from each others and they must be executed in a precise order. On the contrary, since we are in a distributed environment and since we want to perform efficient adaptation, if some actions without logical dependencies can be executed in parallel the schedule must exhibit their potential parallelism. Meanwhile, the execution time taken to implement an adaptation strategy may not be the unique criteria for scheduling. Other factors such as the processor load, the amount of memory or bandwidth used during the adaptation can be of importance. For instance even if it is logically possible to simultaneously transfer two services from one node to another, overloading the network between the two nodes may be incompatible with the quality of service to be preserved for the other activities. As can be easily understood, it is almost impossible to statically define adaptation plans. Too many dynamic factors are involved to make a good planning choice in advance. Therefore our planner is designed to schedule at run time the necessary actions to implement a given strategy.

Run time planning is not an easy task. Thus, we propose to decompose the problem into sub-problems with limited

interactions. Since we consider applications running on a distributed environment, the adaptation that should be realized can impact elements located on different sub-systems and nodes. For that reason we can decompose the planning activity following the location of elements that need adaptation. Distributed planning algorithms have already been proposed in the literature. Georgeff [15], for motion planning in robotic, suggests to use local agents to build local plans and to synchronise them with a supervisor. The decomposition of the main goal into sub-goals is quite simple because each agent manages the behaviour of a distinct robot and only defines its actions. It never acts on other robots. In this case, the supervisor is only needed to find, in the global plan, all actions that can trigger dangerous interactions between robots, for example collisions or the use of the same resource by more than one robot concurrently. Corkill [16] assumes that the goal is only a conjunction of many independent goals and proposes a distributed algorithm based on the NOAH (Nets of Action Hierarchies) framework [17]. Durfee and Lesser [18] propose a partial global planning where each agent communicates with the others during their calculation of local plans. This communication helps them to know the state of the calculation of the global plan. This communication takes time so the agents assume that they know an incomplete, inconsistent and out-of-date state of the global plan. This kind of algorithms degrades the solution that could be found because agents are not always working as a strong collaborative team but it limits the overhead due to communication. Many other algorithms can be found in the literature like [19], [20]. Many of them are based on centralized algorithms extended to distributed problem solving by adding cooperation between agents.

VII. Execution

Once a plan is defined, it has to be executed. As for the previous adaptation functions, execution needs to be efficient due to the dynamism of the environment. Moreover, some characteristics of the SOA or the infrastructure (OS, hardware) where the element to adapt is located have specific requirements. For example, in the service platform OSGi, some dependencies have to be resolved by the platform before starting to migrate services. So execution has to take into account the specificities of the SOA and the infrastructure. That is the reason why we have defined two kinds of actions: abstract and concrete ones. Abstract actions are actions independent from the concrete SOA or infrastructure. They are defined in a *guide* used by the planning phase, making easier the planning of the strategy. Indeed the planner only needs to know information about the current state of the elements, the goal of the adaptation and the available abstract actions. Concrete actions or effectors are the actions dedicated for each service-oriented architecture and environment. Due to the dynamism of the platforms and the environment, we have chosen not to statically define a list of abstract actions but to dynamically specify it from the current concrete actions. Consequently, guides are also dynamically constructed.

A first implementation of our framework has been realised on top of OSGi platforms. OSGi is a specification defining a service-oriented platform and its common services, with a large user base and mature implementations. Applications are built as set of bundles which are library components in OSGi terms. Bundles interact by using or providing services. They import and export Java packages and offer or require services. Services interfaces are Java interfaces. The latest specification defines a way to interconnected several OSGi platforms and allows to bind services running on distributed platforms. OSGi already offers some kind of adaptation actions like the dynamic registration or unregistration of services, but they are not sufficient for the needs of self-organising service-based applications. That's why we define others actions like migration and replication of services.

The migration of service is an environmental adaptation action consisting in moving a service from a node to another. This migration can be useful when a node is overloaded by too many services. If a service X is migrated from an overloaded node A to a node B, resources allocated to this service on the node A are released and can be reallocated for the others. To move a service between platforms, our framework needs to move the bundle because only bundles can register services. Moreover, bundles have dependencies with other bundles according to the services they use and to their imported packages. Nothing has to be done for migration about service dependencies. However as imported packages have to be available where the bundle is, before to move a bundle, our implementation have to solve the packages dependencies. When these dependencies have been resolved, the bundle can be moved from A to B. A bundle is defined by a JAR file. The OSGi platform provides actions to install a bundle with the URL of the JAR file. In our case, before migration, the JAR is accessible on the platform A. A new URL should be created for the platform B. Our framework uses the HTTPService, which is a standard service described in OSGi, in order to publish resource on the network and to build URL for this resource. With this new URL the bundle can be installed. Then, this bundle is started and registers its services on the OSGi platform B. However, not every service have to be registered on the platform B, in our example only the service X. Our implementation provides this possibility by allowing the bundle to ask which services are necessary and the bundle chooses the services to register. In addition, the state of the service is saved before its migration in order to restore it when it is registered on the platform B. Once the service X is registered on the platform B, our framework unregisters X on the platform A and uninstalls the bundle if it does not provide other services or if it is not used to provide packages on A.

VIII. Conclusion

Even if several research works have tackled the problem of software adaptation, now crucial due to the constant evolution of the execution environments, very few consider the heterogeneous and distributed aspects of these environments, as well as the various types of possible adaptations. We propose a generic framework, that, due to its fine grain decomposition into functionalities, can manage different levels of adaptation (service, application, SOA, infrastructure) and cope with dynamically defined adaptation actions for parametric, functional, behavioural, structural, environmental adaptation. We have in particular designed cooperation mechanisms to coordinate distributed analysis and decision, on the fly planning of adaptation actions, using abstract events and abstract actions. Examples of possible specialisations of our framework have been given and a first implementation for OSGi realized. Our current work concerns the implementation on heterogeneous service oriented platforms and on top of cloud infrastructure running an OS capable of virtualisation of resources such as XtreemOS [21].

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

[1] J. Buisson, F. André, and J.-L. Pazat, "Supporting adaptable applications in grid resource management systems," in *8th IEEE/ACM International Conference on Grid Computing*, Austin, USA, 19-21 September 2007.

[2] P.-C. David and T. Ledoux, "An aspect-oriented approach for developing self-adaptive fractal components," in *Software Composition*, ser. Lecture Notes in Computer Science, W. Löwe and M. Südholt, Eds., vol. 4089. Springer Berlin / Heidelberg, 2006, pp. 82–97.

[3] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[4] G. Bastide, A. Seriai, and M. Oussalah, "A self-adaptation of software component structures in ubiquitous environments," in *Proceedings of the 5th international conference on Pervasive services*. ACM, 2008, pp. 173–176.

[5] M. Al-Turkistany, A. Helal, and M. Schmalz, "Adaptive wireless thin-client model for mobile computing," *Wireless Communications & Mobile Computing*, vol. 9, no. 1, pp. 47–59, 2009.

[6] S. Vadhiyar and J. Dongarra, "Self adaptability in grid computing," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 235–257, 2005.

[7] S. Wu and Y.-T. Chang, "A user-centered approach to active replica management in mobile environments," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 11, pp. 1606–1619, Nov. 2006.

[8] "Dynaco Web site," last access on Feb. 2010. [Online]. Available: http://dynaco.gforge.inria.fr/

[9] "OSGi Alliance," last access on Feb. 2010. [Online]. Available: http://www.osgi.org

[10] SCA Consortium, "Building Systems using a Service Oriented Architecture," Whitepaper, 2005. [Online]. Available: http://www.ibm.com/developerworks/library/specification/ws-sca/

[11] E. Cerami, *Web Services Essentials*, S. St.Laurent, Ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002.

[12] P.-C. David and T. Ledoux, "Wildcat: a generic framework for context-aware applications," in *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*. New York, NY, USA: ACM, 2005, pp. 1–7.

[13] "Fipa interacation protocol specifications," last access on Feb. 2010. [Online]. Available: http://www.fipa.org/repository/ips.php3

[14] J. O. Kephart and R. Das, "Achieving self-management via utility functions," *IEEE Internet Computing*, vol. 11, no. 1, pp. 40–48, 2007.

[15] M. P. Georgeff, "Communication and interaction in multi-agent planning." San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988, pp. 200–204.

[16] D. D. Corkill, "Hierarchical planning in a distributed environment," in *IJCAI'79: Proceedings of the 6th international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1979, pp. 168–175.

[17] E. D. Sacerdoti, "A structure for plans and behavior," AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Tech. Rep. 109, Aug 1975.

[18] E. Durfee and V. Lesser, "Predictability versus responsiveness: Coordinating problem solvers in dynamic domains," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988, pp. 66–71.

[19] S. Cammarata, D. McArthur, and R. Steeb, "Strategies of cooperation in distributed problem solving," in *IJCAI'83: Proceedings of the Eighth international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1983, pp. 767–770.

[20] M. Iwen and A. D. Mali, "Distributed graphplan," *Tools with Artificial Intelligence, IEEE International Conference on*, vol. 0, p. 138, 2002.

[21] C. Morin, "Xtreemos: a grid operating system making your computer ready for participating in virtual organizations," *Proceedings of ISORC'07*, vol. 5, pp. 347–368, 2007.